Bachelor of Software Engineering - Game Programming

GD2P02 – Physics Programming

Mathematics Refresher

Overview

- Math Refresher
  - Dot Product, Cross Product
  - Triple Products
  - Planes, Triangles
  - Quaternions, Matrices
- Exercises

GD2P02

Dot Product

- Useful for:
  - Get the angle between vectors.
  - Project a point onto another vector.
- Good operation for parallelization.
- Two vectors in, one scalar out.
  - Sometimes known as the "**Scalar Product**"
- $A \bullet B = A_x B_x + A_y B_y + A_z B_z$
  - Three multiplications and two additions…
- $A \bullet B = |A||B|\cos(\theta)$
  - Theta is the angle between vector A and vector B
  - $|A|$ = magnitude of A, $|B|$ = magnitude of B

Dot Product: Properties

- $A \bullet B = |A||B|\cos(\theta)$

- We should know:
  - For non-unit vector directions, we can determine if they are pointing in the same direction…
    - Based upon the sign of the resulting scalar.
      - Negative means the angle between vectors are more than 90 degrees.
      - Zero means the vectors are at a right angle to one-another.

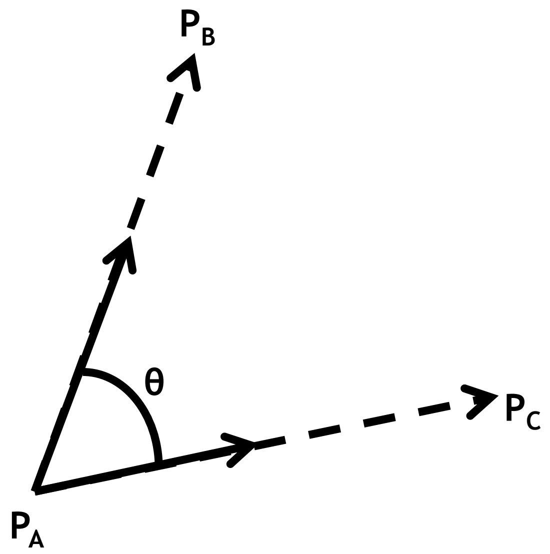
Dot Product: Properties

- $A \bullet B = |A||B|\cos(\theta)$

- Handy to use
  - If only one vector is ***normalised***, then the length to the non-normalised vector is projected onto the result.
  - If both vectors are ***normalised***, then we can calculate the angle between them.

Dot Product: Finding the angle

$\cos \Theta = \text{dot}(|P_B - P_A|, |P_C - P_A|)$



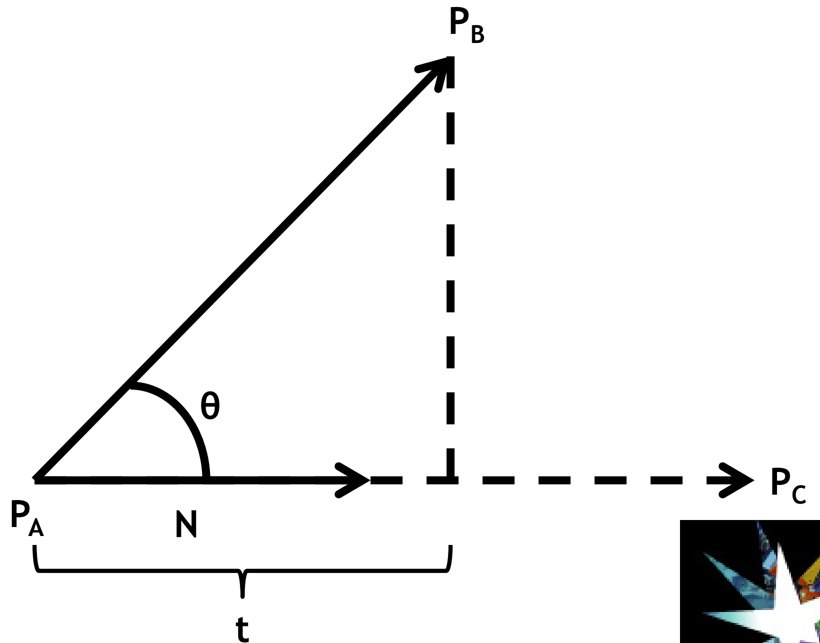
Dot Product: Projection

- Projection:
  - If one of the vectors is not normalised, then the dot product will **project the unnormalised vector onto the normalised vector**…

N = normalised(P_C - P_A)

t = dot(P_B - P_A, N)

- The dot product measures the length of the projection… t



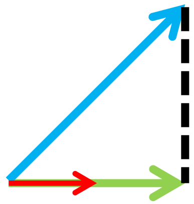
GD2P02

Dot Product: Projection

- Green = g
- Blue = b
- Red = g dot b

Dot Product: Projection and vectors in projection

- Red = normalised g
- Blue = b
- Green = b parallel
- Black = b perpendicular



- b perpendicular + b parallel = b

Dot Product: Code Example:

```
CVector3 a( 0.0f, 0.0f, 1.0f );
CVector3 b( 0.0f, 1.0f, 0.0f );

float d = CVector3::Dot( a, b );

float fAngle = acos( d );

// fAngle will be 1.57 radians
// 1.57radians is 90 degrees
```

Cross Product

- Useful for:
  - Perpendicular vector of two non-parallel vectors.
- Sometimes known as the “vector product”
- Remember the cross product returns a vector!
  - Resulting vector is perpendicular to the two input vectors.
  - Two input vectors form a triangle…
    - Resulting vector is the direction of the face of the triangle.
      - At 90 degrees from an edge of the triangle.
- Example:
  - One vector: Along the x-axis, Another: Along the y-axis
  - Result: vector along the z-axis!

Cross Product

- $A \times B = (A_yB_z - A_zB_y), (A_zB_x - A_xB_z), (A_xB_y - A_yB_x)$
- $A \times B = -B \times A$
- $|A \times B| = |A||B| \sin(\theta)$

- Order of multiplication is important!
- $A \times B \neq B \times A$
  - Opposite order results in a vector pointing in the opposite direction!

Cross Product: Code Example

```
inline CVector3 CVector3::CrossProduct(
                                      const CVector3& v0,
                                      const CVector3& v1)
{
    return CVector3(v0.y * v1.z - v0.z * v1.y,
                    v0.z * v1.x - v0.x * v1.z,
                    v0.x * v1.y - v0.y * v1.x)
           ;
}
```

Cross Product: Cross Product Matrix

- Remember that multiplying a vector by a matrix results in a vector…
- Create a "Cross Product Matrix"
  - Multiply a vector by the matrix, the result is the same as performing the cross product…

$A \times B = \hat{a}B$

$$\hat{a} = \begin{bmatrix} 0 & -A_z & A_y \\ A_z & 0 & -A_x \\ -A_y & A_x & 0 \end{bmatrix}$$

Cross Product: Scalar Triple Product

- Also known as: Mixed or Box Product
  - Measures the volume of the parallelepiped bounded by three vectors, A, B, and C.

$$s = A \bullet (B \times C)$$

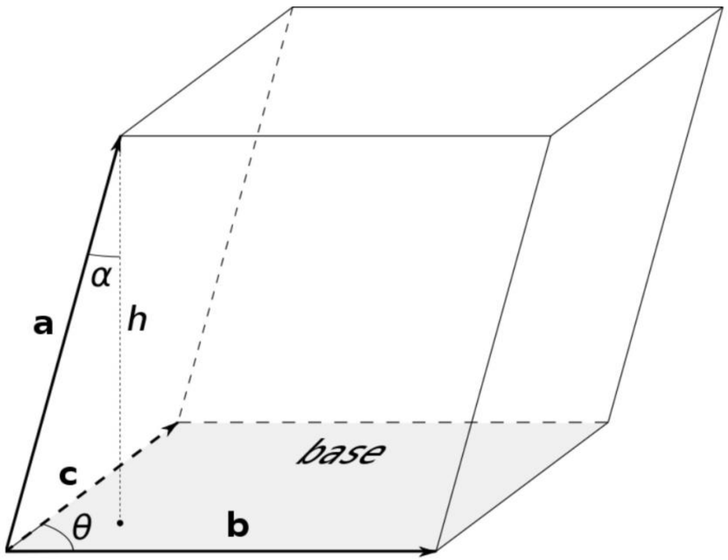


Fig 1: Parallelepiped from wiki.

Therefore:

$$A \bullet (B \times C) = B \bullet (C \times A) = C \bullet (A \times B)$$

Cross Product: Vector Triple Product

- Useful for creating an orthogonal basis from linearly independent vectors.
- Result is a vector.

  v = A x (B x C)

- Example Basis:

  B, B x C, and B x (B x C)

Let: B = < 1, 0, 0 >, C = < 0, 1, 0 >

- Therefore:

  B = < 1, 0, 0 >

  B x C = (< 1, 0, 0 >) x (< 0, 1, 0 >) = < 0, 0, 1 >

  B x (B x C) = B x (< 0, 0, 1 >) = < 0, -1, 0 >

Cross Product: Vector Triple Product continued…

- Triple Product Expansion:

$$A \times (B \times C) = (A \bullet C)B - (A \bullet B)C$$

- Also known as Lagrange's Formula.

- This relationship is useful for rigid body dynamics and geometric algorithms.

Cross Product: Plane Equation

- A point can be on either side of a plane.
    - Rejection test
- Plane: Cuts the world in two!
    - Combine multiple planes to slice the world into smaller and smaller chunks!
- The plane equation allows for deciding which side of the plane we are on…
    - Or if with multiple planes, which chunk we are in!
- A triangle is on a plane…
    - Which side of the plane a point is on?
    - How far away from the plane is the point?

Cross Product: Plane Equation continued…

- To define a plane:
  1) Need a point on the plane.
  2) Normal of the plane

- Plane Equation:

  d = Ax + By + Cz

  - < A, B, C > is the plane normal.
  - d is the shortest distance from the plane to the origin.
  - < x, y, z > represent coordinates of the point on the plane…
  - N_p is the normal, P_p is a point on the plane

    $d = N_p \bullet P_p$

Cross Product: Plane Equation continued…

- Which side of the plane is a point on?

1. Calculate the reference distance d for the plane…

   d = planeNormal • planePoint

2. Next, check the point in space…

   val = (planeNormal • pointToCheck) – d

3. If:
   - val is Zero: pointToCheck is on the plane!
   - val is > Zero: pointToCheck is in-front of the plane!
   - val is < Zero: pointToCheck is behind the plane!

Cross Product: Line-Plane Intersection

- Line Segment vs Plane:
  - Test each end of the line against the plane.
    - If each end is on alternate sides of the plane…
    - Then there has been an intersection!
      - Then calculate the exact point of intersection!
  - Known:
    - Equation of the plane.
    - Equation of the line.
    - Combine them to determine the intersection point!

Cross Product: Line-Plane Intersection continued…

- Line Segment vs Plane:
  - Line segment:
    - p(t) = p0 + t(p1 – p0)
    - p0 is the line start point.
    - p1 is the line end point.
    - Scalar t goes from 0 to 1.
    - p(t) is a point on the line!
  - Insert the line equation into the plane equation:
    - Dot(planeNormal, pointOnPlane – anyPoint) = 0
    - Dot(planeNormal, (p0 + t(p1 – p0)) – anyPoint) = 0
  - Rearrange to make t the subject:
  - t = dot(planeNormal, pointOnPlane – p0) / dot(planeNormal, p1 – p0)

Cross Product: Line-Plane Intersection continued…

- Line Segment vs Plane:
  - t = dot(planeNormal, pointOnPlane – p0) / dot(planeNormal, p1 – p0)
  - If:
    - t > 0 and t < 1: Intersection has occurred between the end points.
    - t == 0: Intersection at first end point.
    - t == 1: Intersection at second end point.
    - t > 1: Intersection beyond second end point.
    - t < 0: Intersection before first end point.

Cross Product: Line-Plane Intersection continued…

- Line Segment vs Plane:
  - Triangle is three line-segments.

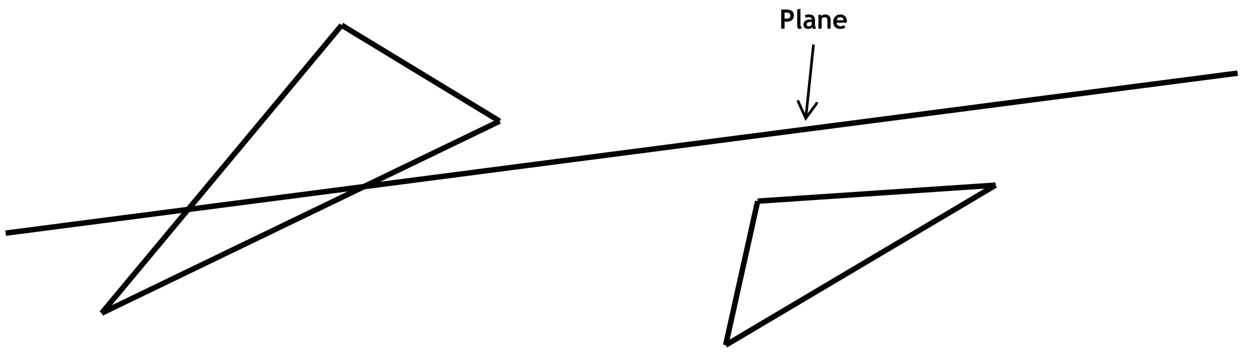


Fig 1: Top down view of a plane and two triangles, one triangle intersecting the plane, the other triangle not intersecting.

Cross Product: Slicing or Cutting Triangles

- Cut a triangle with a plane (or line in 2D)…
- Possible outcomes:
  - Points are all on one side
  - Points are all on other side
  - Plane cuts, producing three triangles from the cut.
    - Two points one side, one point the other…

Cross Product: Slicing or Cutting Triangles continued…

- Two points one side, one point the other; this requires new triangles!

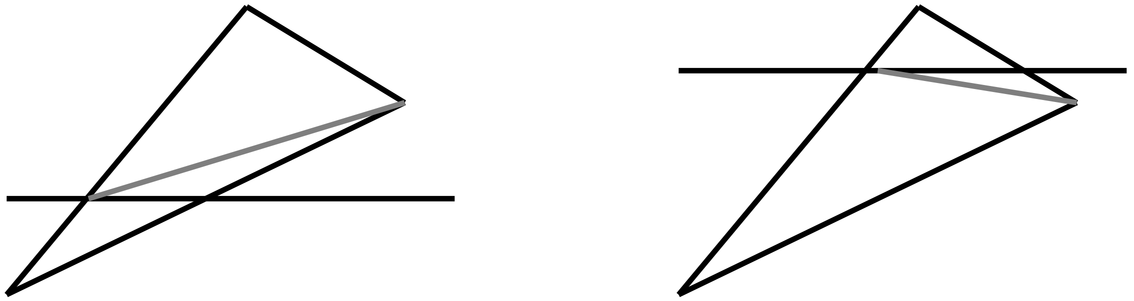


Fig 2: Triangle Cutting.

Cross Product: Slicing or Cutting Triangles continued…

- Algorithm:
  - Two empty vertex lists:
    - Vertices above the plane, Vertices below the plane.
  - After the calculation, if there was an intersection:
    - One list has four vertices, The other has three vertices.
  - Start at a random vertex.
  - Go either clockwise or anti-clockwise around the triangle…
  - Check each line segment against the plane.
  - If the start and end vertex are on different sides of the plane:
    - We have a cut, so split the line, add the new vertex to both lists.

Cross Product: Slicing or Cutting Triangles continued…

- Algorithm continued:
  - If there was no intersection, either the above or the below array will have three vertices, the other zero.

Quaternions

- Useful for:
  - SLERP
  - Avoiding gimbal lock
  - Representing angle and rotations
  - Better than matrices for representing orientations
    - Memory efficient!
- Physics requires quaternions!
  - Four-dimensional hyper-plane!
  - Multiple dimension complex numbers!
- $q = w + (xi + yj + zk)$

Quaternions Rules:

- Remember:
  - $i^2 = j^2 = k^2 = -1$
  - $ij = -ji = k$
  - $jk = -kj = i$
  - $ki = -ik = j$
- Conjugate: inverse of unit quaternion
  - $q^{-1} = w - xi - yj - zk$
- Quaternion multiplication is NOT commutative.
  - $q_1 = w_1 + v_1$, where $v_1 = \langle x, y, z \rangle$
  - $q_1 * q_2$; $w = w_1 w_2 - v_1 \bullet v_2$

    $v = w_1 v_2 + w_2 v_1 + v_1 \times v_2$

Quaternions continued…

- Normalising:
  - Magnitude is 1.
  - $\mathrm{Sqrt}(w^2 + x^2 + y^2 + z^2) = 1$
- Rotation mapped to quaternion:

$$q = \cos(\theta/2) + x(\sin(\theta/2)) + y(\sin(\theta/2)) + z(\sin(\theta/2))$$

- Convert a vector to a quaternion:

$$q(v) = < 0, v_x, v_y, v_z >$$

- Rotate arbitrary point v by applying quaternion rotation :
  - $v' = qvq^{-1}$
  - v' is the rotated point!

MEDIA DESIGN SCHOOL GAME DEV

Quaternions continued…

– Quaternion to Rotation Matrix

$$
\begin{bmatrix} 1 - 2(q_y^2 + q_z^2) & 2(q_x q_y - q_s q_z) & 2(q_x q_z - q_s q_y) \end{bmatrix}
$$
$$
\begin{bmatrix} 2(q_x q_y - q_s q_z) & 1 - 2(q_x^2 + q_z^2) & 2(q_y q_z - q_s q_y) \end{bmatrix}
$$
$$
\begin{bmatrix} 2(q_x q_z - q_s q_y) & 2(q_y q_z - q_s q_x) & 1 - 2(q_x^2 + q_y^2) \end{bmatrix}
$$

MEDIA DESIGN SCHOOL GAME DEV

Quaternions continued…

- Memory optimisation:
  - Embed the scalar component into the vector part of the quaternion…
    - Requires the quaternion to be a unit length…
    - Extracting the scalar part becomes additional cost…
      - Trading memory for speed!
    - $q_s = sqrt(1 – q_x^2 + q_y^2 + q_z^2)$
    - Ensure q_s is positive, to store the quat in three floats… (not four!)
    - Perhaps overkill… remember to make code work first…
      - Before optimising!

Matrices

– $Q_{nxm} = Q_{2x3} = \begin{bmatrix} a & b & c \\ d & e & f \end{bmatrix}$

– n rows, m columns

– Remember 4 by 4 matrices…

– Decompose a matrix: Extract each of the following:

  - Rotation
  - Translation
  - Scaling

Matrices continued…

- Beware of drift…
- Check rotation matrix's columns are orthogonal to each other…
  - Use the cross product to re-orthogonalise them!
- Multiplication:
  - The column of the first matrix has to match the size of the row of the second matrix…
  - $Q_{nxm} \times S_{mxp} = R_{nxp}$
- Large matrix operations are ideal for parallelization.

Matrices continued…

- Inverses
  - 4 by 4: costly…
  - Pure rotation matrix:
    - No reflection, no scaling.
    - Then the inverse is the same as the transpose!
  - A matrix is non-singular and invertible only if:
    - The determinate is non-zero.
  - If a matrix has a determinate of zero:
    - It is a singular matrix that is non-invertable.
  - Singular Matrix: Square matrix that does not have a matrix inverse.
    - Singular if and only if its determinate is 0.

Matrices continued…

- Inverses
  - Zero determinant: When?
    - Two rows or columns are equal.
    - An entire row is zero.
    - A row or column is a multiple of another row or column.

- Physics will require:
  - Identity matrices.
  - Rotational matrices.
  - Orthogonal matrices.
  - Scale and translation matrices.
  - Matrix multiplication!

Exercise

- Exercise 001.1 – Lagrange's Formula
  - Confirm Lagrange's formula is correct.
    - The Triple Product Expansion…
  - Create a small C++ project…
  - Calculate the RHS and the LHS for different test cases.

- Exercise 001.2 – Plane vs Point Function
  - Plane: Defined by a point on the plane and a normal.
  - Point: 3D Point in space.
  - Create a C++ function to collide a point vs a plane.
  - Return the result: ON_PLANE, INFRONT, or BEHIND.

Exercise

- Exercise 001.3 – Line Segment vs Plane Function:
  - Plane: Defined by a point on the plane and a normal.
  - Line Segment: Two points, one for each end.
  - Create a C++ function to collide a line segment vs a plane.
  - Return the result: TRUE or FALSE.
    - TRUE = collision occurred, FALSE = no collision.

- Exercise 001.4 – Triangle vs Plane Function
  - Plane: Defined by a point on the plane and a normal.
  - Triangle: Defined by three points.
  - Create a C++ function to collide a triangle vs a plane.
  - Return the result: TRUE or FALSE.

Exercise

- Exercise 001.5 - Triangle Cutter:
  - Implement an application that allows the user to:
    - Create a triangle:
      - By clicking three points to form the triangle.
      - T key resets the triangle…
    - Create a line:
      - By clicking two points to form the line segment.
      - L key resets the line…
    - If the line intersects the triangle…
      - Then render the resulting triangles
      - Use colour to make the triangles obvious…

Summary

- Math Refresher
  - Dot Product, Cross Product
  - Triple Products
  - Planes, Triangles
  - Quaternions, Matrices
- Exercises